

# A Software Organization Platform (SOP)

Sebastian Weber<sup>1</sup>, Ludger Thomas<sup>1</sup>, Ove Armbrust<sup>1</sup>, Eric Ras<sup>1</sup>, Jörg Rech<sup>1</sup>, Özgür Uenalan<sup>1</sup>, Martin Wessner<sup>1</sup>, Marcel Linnenfelser<sup>2</sup>, and Björn Decker<sup>3</sup>

<sup>1</sup> Fraunhofer IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany  
{Forename.Surname}@iese.fraunhofer.de

<sup>2</sup> Synflag Online Agentur, Wendelinusstraße 38, 67435 Neustadt, Germany  
m.linnen@synflag.com

<sup>3</sup> empolis GmbH, Europaallee 10, 67663 Kaiserslautern, Germany  
Bjoern.Decker@empolis.com

**Abstract.** Software engineering is a highly collaborative and complex process involving a large number of different roles. In this process, every activity comprises the development of software artifacts that are created by a specific role or provided by others and that might serve as input for subsequent activities. Hence, software engineering highly depends on efficient management of and access to various kinds of information. This article presents the concept of a Software Organization Platform (SOP), which aims at supporting the creation, storage, and exchange of such information as well as personal or organizational experiences by using less-formal, innovative Web 2.0 technologies.

**Keywords:** Software Organization Platform, Software Engineering Environment, Web 2.0, Software Lifecycle, Knowledge Management, Learning Software Organization, Experience Factory, Collaboration, Communication, Wiki, SME

## 1 Introduction

Demands for higher software quality are increasing steadily, while development cycles are becoming shorter and shorter. Furthermore, in many cases, software development is done by highly specialized teams around the globe that communicate using state-of-the-art technology. For the efficient production of high-quality software under those conditions, software engineering is gaining more and more importance. Effective software engineering requires efficient communication between the involved roles as well as well-established processes. Engineering software means going through many phases, such as requirements engineering, architecture and design, implementation, testing, and deployment, which need constant coordination and collaboration. There exists a high correlation between the different phases, the roles involved, the artifacts created, etc. An artifact of a particular phase (e.g., an UML class diagram for business analysis) can be the basis of one or more artifacts of other

phases (e.g., several UML diagrams during design) and a single failure in one of the early phases can cause enormous efforts when detected late in the development process. In modern software engineering, the efficient creation, storage, and exchange of artifacts and information are major challenges.

Besides the exchange of project-related information and artifacts, identifying, capturing, and delivering explicit, tacit, organizational, or personal knowledge and experiences of different maturity (on demand or proactively) is a challenge for any software organization [1]. Every phase of the development process should benefit from experiences and empirical evidences, such as personal experiences of the stakeholders (e.g., in project VZ, feature XY was not accepted very well), as well as best practices (e.g., design pattern XY reduced the failure rate by 50%), or worst practices (e.g., code smells). Modern Learning Software Organizations (LSO) need to provide usable mechanisms for handling such information and knowledge in globally operating software developing organizations.

This article presents the concepts and current status, as well as the vision of a so-called Software Organization Platform (SOP), which has been prototypically developed at Fraunhofer IESE on the basis of Web 2.0 technologies. The concept of an SOP especially targets small and medium-sized organizations on low to medium maturity levels that need usable, lightweight solutions for the intra-organizational management of software engineering related knowledge and information. It is based on core aspects of Learning Software Organization (LSO) and extends them by utilizing the concepts of a Software Engineering Environment (SEE) in combination with lightweight knowledge management support. Intra-organizational collaboration as well as collaboration across organizational boundaries are an integral part of an SOP. Therefore, SMEs (small and medium-sized enterprises) utilizing agile software development benefit from an SOP.

Decker et al. first introduced the concept of an SOP that can handle the exchange of information and collaborative work on software artifacts [2]. Their initial prototype (a collection of extensions of a Wiki) showed for the phase of requirements engineering that an SOP can provide significant benefits and support for collaborative information exchange and communication between the involved personnel. They also identified support for collaboration between all kinds of stakeholders and flexibility as being the major strengths of such a platform.

However, their initial implementation contains several weaknesses, such as development limitations for embedded CASE tools. In addition, the platform cannot deliver rich user experiences, which users expect from such a platform. From the developer's point of view, the class of potential applications is limited (e.g., interactive visualization tools are difficult to implement).

In order to implement the concept of an SOP, all information emerging throughout a software project should be made accessible to the relevant roles in the right way and at the right time. For the aggregation of such information, an SOP needs a central repository of interlinked information, knowledge, and experiences in one information space, i.e., in a single, uniform, and accessible platform.

Due to the positive results from the initial case studies where our initial prototype was used, the concept of an SOP seems to be suitable for extending the view from support for specific phases to collaborative development during the entire software

lifecycle. With this experience, we define the conceptual requirements for such an advanced development environment:

- (RqColl): An SOP supports *collaboration* and the easy exchange of information between the roles in the software lifecycle.
- (RqLife): An SOP supports the whole software *lifecycle*, not only selected phases and specific tasks.
- (RqIntg): An SOP supports the *integration* of data and information from various continuously updated sources within the software organization. It integrates a broad range of specialized external CASE tools that are used during the development process. It supports versioned, concurrent, and non-destructive editing of information.
- (RqLink): An SOP allows semantically *linking* information, people, and software artifacts. The system should facilitate knowledge acquisition by collecting semantic information either automatically or without much effort.
- (RqSEF): An SOP is *scalable*, *extendable*, and *flexible*. It can be adapted to a broad range of organizations. It allows users to extend or adapt it to their personal needs. It provides views for various stakeholders and devices.
- (RqOpen): An SOP is *open* for its users and provides non-restrictive, non-formalized access to information. It supports the use of open standards and interoperates with external tools using those standards.
- (RqInfo): An SOP reduces the *information overload* knowledge workers are exposed to every day. It filters and aggregates relevant information and provides useful visualizations.

The defined requirements neither insist on being universal nor do they have a broad empirical basis. They have been set up as heuristics on the basis of existing personal experiences with the initial prototype. However, they serve as conceptual guidelines for the analysis and creation of an SOP and are referred to throughout this article.

The initial prototype of an SOP (throughout this article, we refer to it as “SOP 1.0”) helped us to define the underlying concepts and identify the requirements of an SOP that constitutes a consequent advancement of an SEE focusing on knowledge management and collaboration. We decided to develop a new implementation of an SOP (throughout this article, we refer to it as “SOP 2.0”), which is based on a sophisticated architecture that enables us to fulfill the defined requirements. Consequently, “SOP 1.0” and “SOP 2.0” refer to concrete SOP implementations, while the term “SOP” refers to the concept.

In the following, this article presents related work that supports some of the requirements mentioned above. Later, it also illustrates the conceptual foundation in the Web 2.0 paradigm and the architecture of our implementation of an SOP (SOP 2.0) that is currently under development. Recent experiences and the possible roles of our new SOP implementation regarding selected phases of the software lifecycle are also addressed. Finally, a more general view on the status of our new SOP implementation and future work is described.

## 2 Related Work

The term Software Engineering Environment (SEE) emerged during the time of the so-called software crisis [3]. The core idea is a platform that supports all tasks for developing, reengineering, and maintaining complex software systems [12] by integrating tools for the creation of the actual software artifacts with tools for the acquisition, storage, and presentation of knowledge, plus collaborative tools for coping with the immense need for communication within a software project.

CODE is a knowledge-based tool developed by the Artificial Intelligence Laboratory at the University of Ottawa in the late 1990s. It was created for capturing, editing, and documenting knowledge within a software project [4]. The software was intended to replace conventional tools (e.g., word processors) which, from today's perspective, actually did not work. Nevertheless, key concepts of CODE (e.g., the use of ontologies) are interesting and can still be seen as state of the art [5]. Henninger [6] introduced a similar tool, BORE, which aims at collecting and managing software development knowledge. BORE supports evolving knowledge through case-based techniques and domain analysis methods that capture emerging knowledge and synthesize it into generally applicable forms.

Both the CODE and the BORE system rely on formalizing the users' input in order to acquire knowledge and make it more accessible with database queries, etc. Apart from pilot usages and proof of concept studies, it seems that such approaches seem to have little impact on practice [5]. One reason might be that people tend not to spend much extra effort on formalizing knowledge without deriving direct benefits, as reported by Tim O'Reilly [7]. He states that users try to avoid extra effort for providing structure or semantics because there are no immediate benefits visible to them. The broad acceptance of lightweight Web 2.0 technology such as tagging rather than highly sophisticated and formalized approaches (Semantic Web techniques) underpins this opinion [8].

In the domain of experience management, which is a subdomain of knowledge management, the paradigm of the Experience Factory (EF) has been developed [9]. EF is an infrastructure designed to support experience management in software organizations. It supports the collection, pre-processing, analysis, and dissemination of experiences and represents the physical or at least logical separation of the project and experience organization. This separation is meant to relieve the project teams from the burden of finding and preserving valuable new experiences that might be reused in later projects. Similar to the aforementioned knowledge management tools BORE and CODE, EF prototypes from academia as well as commercial products seem to rely mainly on formal and heavyweight technologies.

In recent years, several key concepts of SEEs have been integrated into IDEs (Integrated Development Environments) such as Eclipse, NetBeans, or Microsoft Visual Studio. Those applications focus mainly on specific phases of software development, such as design and implementation, rather than supporting the whole software lifecycle. Additionally, the collaborative aspects of software development in globally distributed teams, the increased importance of development processes, as well as the storage and exchange of various kinds of knowledge are not well supported in current IDEs.

The Jazz Platform that is currently being developed at IBM is a commercial project aimed at providing a scalable platform for the deep integration of a broad range of tasks across the software lifecycle [10]. It incorporates collaborative tools into the Eclipse IDE to help developers interact with each other and make distributed software teams more productive. According to Randall Frost [10], this approach requires that people, teams, and organizations share Jazz's style of collaboration and that all use the Eclipse platform. Similar software systems are currently offered by Polarion and Microsoft [11]. However, single tool (suite) solutions might be problematic in terms of embracement because different stakeholders have diverse preferences regarding the tools they use for creating their assets or for communication. Products such as Jazz aim at developers, and non-technical stakeholders are not considered or overstrained. In addition, industry experience with system development has shown that single tool solutions cannot effectively improve the entire development process [12].

In recent years, communication has become more and more important during the software development process, mainly due to globally distributed working teams and the large number of different stakeholders involved. However, modern SEEs do not yet provide enough support for collaborative work. On the other hand, Web-based concepts and technologies, such as emailing, instant messaging, or Wikis have been embraced by all stakeholders working on a software project. It has been shown that Wikis are successful collaboration tools [13]. Harnessing collective intelligence, which is one of the key concepts of Web 2.0, for capturing knowledge can be achieved through Wikis (e.g., Wikipedia [14]).

The FLOSSs (Free/Libre Open Source Software) Trac is an enhanced Wiki with issue and bug tracking functionality [15]. Nevertheless, Trac is only adequate for small software projects without software processes because it only addresses the implementation phase by displaying code from a SVN repository and project management in terms of roadmap and timeline support. Other collaborative FLOSSs (e.g., SnipSnap, MASE, SubWiki, eclipseWiki [16]) are also limited to just a few aspects of a software lifecycle.

### **3 SOP 2.0 – Implementation of the SOP Concept**

The main motivation of an SOP is to provide integrated access to information, experiences, and artifacts within an LSO. Thus, we define an SOP as a lightweight SEE with knowledge management capabilities for establishing an LSO. In contrast to the more formalized solutions briefly described above, an SOP leverages the embraced Web technologies to provide an open, lightweight, and collaborative platform. A key concept of an SOP is CASE tool integration – an SOP might offer a way for creating and editing specific software artifacts within a Wiki (e.g., requirements), while artifacts that should not be edited inside the Wiki (e.g., source code) can be referenced by some uniform integration mechanism (e.g., subversion access). An SOP does not constrain its users to one “silver bullet” tool but allows them to use their preferred tools, such as a specific CASE tool for coding (e.g., Eclipse) or a specific collaboration tool (e.g., email client, instant messenger). Our goal of an SOP is to integrate such tools in the sense that information is gathered, stored, and preprocessed semanti-

cally, even if it is not provided by using an SOP (i.e., an embedded application within an SOP).

In contrast to most of the related work, an SOP relies on lightweight Web 2.0 concepts (e.g., tagging) and technologies (e.g., Wiki) that do not constitute a perfect general solution but can be adapted to the specific needs of the users (e.g., tagging as basic semantic technology vs. sophisticated Semantic Web technologies) [8]. However, an SOP pursues the approach of implicitly creating semantic relationships and acquiring knowledge either automatically or without much effort (e.g., by using templates or tagging). Worldwide collaboration is an integral part of the Web 2.0 spirit, thus SOPs harness appropriate techniques, such as syndication, recommendation, collective intelligence, mashups, or SaaS (Software as a Service). An SOP offers a flexible and pragmatic platform that can be integrated with existing (and embraced) specialized tools suitable for every group of stakeholders. The target audience of an SOP are SMEs that leverage an agile development approach.

Section 3.1 describes the Web 2.0 concepts, which are an integral part of an SOP. Section 3.2 describes the architecture of our new implementation of SOP (referred to as SOP 2.0).

### 3.1 The Web 2.0 Paradigm as the Fundamental Basis of an SOP

This section describes how Web 2.0 technologies as a fundamental basis for a platform have the potential to fulfill the defined SOP requirements. Web 2.0 is not only a special technology, but also an umbrella term that refers to a class of Web-based applications [17]. In the spirit of Web 2.0, Web-based applications make the most of the intrinsic advantages of a platform, get better as more people use them by capturing network effects, harness collective intelligence through user-generated contents, enable collaborative work, deliver rich user experiences via desktop-like interfaces, and combine data from multiple sources into new services [18].

Wiki technology enables users to easily create, edit, and link documents (*RqSEF*). Additionally, semantic annotations and typified links enable the creation of ontologies (e.g., with the Semantic MediaWiki<sup>1</sup> extension) (*RqLink*). Wikis in general facilitate communication through a basic set of features and delegate the actual method of coordination to the people who are using the Wiki (*RqColl*). Such basic features are:

- *One place publishing*, meaning there is only one version of a document that is regarded as the current version (*RqIntg*).
- *Simple and safe collaboration*, referring to versioning and locking mechanisms that most Wikis provide (*RqIntg*).
- *Easy linking*, meaning that documents within a Wiki can be linked by their title using a simple markup – which is important for coping with the intertwined nature of software artifacts (*RqLink*).
- *Description on demand*, meaning that links can be defined to pages that have not been created yet, but might be filled with content in the future.
- *No / low cost* opportunity for capturing software artifacts, since most Wikis are open source – like components used in an SOP.

---

<sup>1</sup> <http://www.semantic-mediawiki.org>

Harnessing collective intelligence is another important concept of Web 2.0. Collective intelligence means combining the behavior, preferences, or ideas of a group of people to create novel insights (*RqColl*) [18]. Collecting answers from a large group of people enables decision-makers to draw statistical conclusions about the group that no individual member would have known by himself. As more people participate, chances increase that someone will provide information regarding specific topics, correct mistakes and improve information quality, and show interest in the contributions made. Wikipedia is one prominent example of leveraging collective intelligence. As MediaWiki is the basis of Wikipedia, our SOP implementation also has the potential of establishing collective intelligence within a software company.

Syndication is another concept of Web 2.0 that enables people to cope with information overload (*RqInfo*). Feeds in combination with presentation mashup platforms (i.e., Web desktops), such as Netvibes<sup>2</sup>, are established Web 2.0 features designed to reduce information overload.

### 3.2 Architecture of SOP 2.0

Our original implementation of an SOP (i.e., Sop 1.0) is mainly built on MediaWiki, which is a popular Wiki platform used by various kinds of collaborative Web 2.0 services, such as Wikipedia, Wikibooks, or Wikiversity. First evaluations of SOP 1.0 in the context of public and industrial projects (see next section) showed that the developed platform is well accepted, but that the underlying Wiki and the Wiki syntax are not sufficient for providing an easy-to-use and easy-to-understand UI that is suitable for adequately supporting a software lifecycle. In addition, the implementation did not comply with the requirements defined above.

Thus, we started to develop an improved version of an SOP implementation (SOP 2.0) with the goal of fulfilling the defined requirements and establishing an SEE that supports all stakeholders throughout the entire software development process.

Table 1 illustrates how the two versions fulfill the requirements of an SOP. Partly-filled circles mean that this platform is only applicable to a limited extend in terms of complying with the requirement.

**Table 1. COMPLIANCE WITH THE REQUIREMENTS OF AN SOP (REFER SECTION 1)**

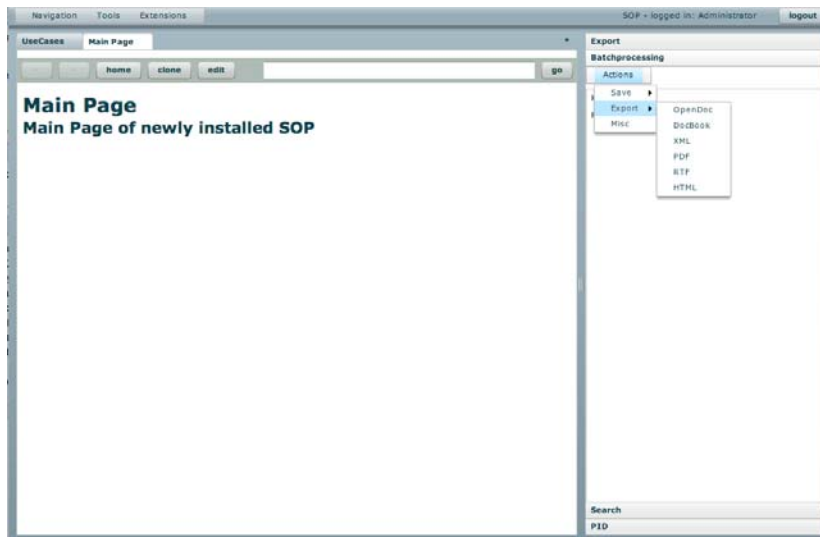
	RqColl	RQLife	RqIntg	RqLink	RqSEF	RqOpen	RqInfo
SOP 1.0 (MediaWiki-based)	●	◐	◐	●	◐	◐	◐
SOP 2.0 (Framework / Flex)	●	●	●	●	●	●	●

Because SOP 1.0 mainly consists of the MediaWiki application with the Semantic MediaWiki extension installed and a set of plug-ins, the development of applications for SOP 1.0 is constricted to the possibilities of MediaWiki's special pages. This

<sup>2</sup> <http://www.netvibes.com>

reduces flexibility and extensibility for developers (*RqSEF*). It is also not possible to enable a user to customize SOP 1.0 to his preferences. These restrictions influence and restrict all requirements marked by partly-filled circles. MediaWiki only provides basic mechanisms (e.g., watchlists) for reducing information overload (*RqInfo*). Because SOP 1.0 is limited regarding advanced user interfaces and sophisticated interactive visualization, it is difficult to support all phases of the software lifecycle (*RqLife*). MediaWiki is not designed to fulfill the requirements *RqOpen* and *RqIntg*, because, by its very nature, it does not support integration of data or complex applications into the platform.

Especially for providing a rich user experience and a more flexible and customizable UI, the idea was born to implement SOP 2.0, which features a GUI using Adobe's Flex technology<sup>3</sup>. This approach combines the benefits of MediaWiki (and therefore includes SOP 1.0 with all its benefits) and the Wiki-based way of managing information, as well as the benefits of a desktop-like UI.



**Fig. 1.** SOP 2.0 User Interface

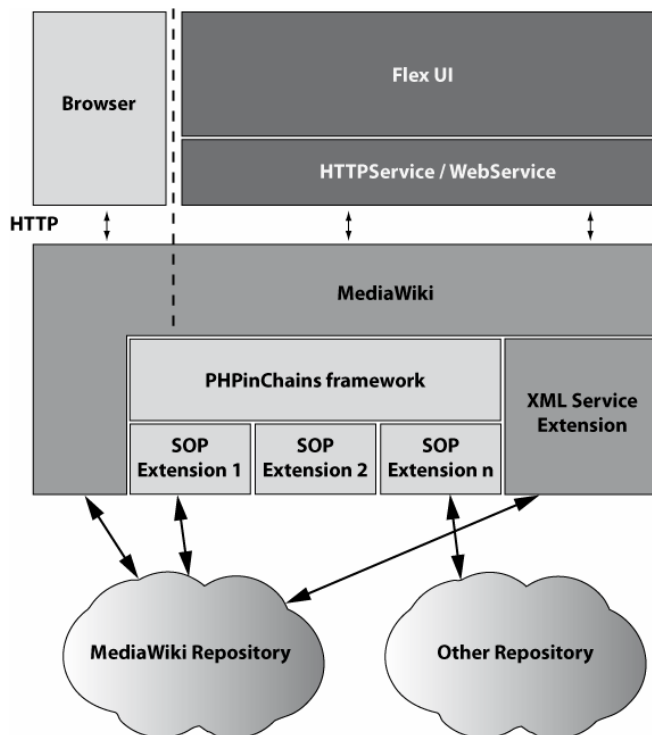
The Flex UI is built around a browser control mechanism (i.e., a widget or UI component in the context of Flex), which allows browsing and editing Wiki articles (see left side of Fig. 1). Additionally, context-sensitive data and other controls (e.g., drop-down menus) as well as often-needed functionalities can be provided in an accordion control (see right side of Fig. 1).

To assist and support people in creating and editing articles, SOP 2.0 provides an integrated template mechanism. This graphical template mechanism offers the facility to create different forms for different types of articles, thus providing implicit structure and metadata for an article. Thus, SOP 2.0 fulfills the goal of an SOP, namely

<sup>3</sup> <http://www.adobe.com/products/flex/>

that the user provides semantic information with no or low additional effort enabled by low threshold technologies (e.g., tagging).

Fig. 2 provides a high-level view on the architecture of SOP 2.0. SOP 2.0 features the ability to build so-called hybrid extensions (provided by PHPinChains<sup>4</sup> Web framework), which may provide an HTML UI (accessible without a Flash plug-in) as well as a Flex UI (needs Flash plug-in). In addition to the Flex UI, SOP 2.0 enables developers to create alternate UIs for devices, e.g., Java ME applications on cell phones, using the XML Service Extension already used by the Flex UI. This might be interesting for stakeholders who are often away on business trips (e.g., accessing information via a mobile phone).



**Fig. 2.** Architecture of SOP 2.0

Table 1 illustrates that SOP 2.0 has the potential of fulfilling all the requirements demanded of an SOP. The new architecture of SOP 2.0 enables the implementation of advanced applications that are not or only difficult to realize with SOP 1.0 (*RqSEF*). Sophisticated applications that are similar to desktop applications are possible, for example, for visualizing relationships (interactive UI) between different pieces of information within the Wiki (*RqInfo*). The architecture enables integration of CASE tools through proprietary implementations or by leveraging external APIs for every phase of the software development cycle (*RqLife*). In respect to SOP 1.0, the new

<sup>4</sup> See <http://phpinchains.synflag.com> for a framework overview.

version is subject to fewer restrictions in designing interfaces, thus developers are supported in creating more user-friendly and more consistently designed user guidance (*RqSEF*). SOP 2.0 allows embedding arbitrary complex applications, and its underlying architecture is designed to integrate external information (*RqIntg*). In addition, it leverages the embraced Web 2.0 technologies (*RqOpen*).

## 4 Experiences and Visions

In the course of the following sections, we will describe the experiences we made with SOP 1.0 – our initial implementation of an SOP. Scenarios 1 – 2 provide concrete experiences for particular phases of a typical software process. We explain which of the activities are currently supported by our implementation and which are going to be supported in the future. Scenarios 3, 4, and 5 describe the application in the area of process modeling, experience management, and individual learning, which are phase-comprehensive examples. For some of the activities, this section provides scenarios from selected case studies.

### 4.1 Scenario 1: Requirements Engineering

A recent study shows that the requirements are often captured in office documents [20], which are shared via email or a collaboration platform. However, this approach has several shortcomings [2]: *Collaboration chaos* due to concurrent changes by different stakeholders or late exchange via email, or *distribution chaos* if requirements are distributed across several documents; a high risk that links between the documents will break; *untyped links* (semantics of links cannot be captured); and finally, *no explicit versioning and baselining* of requirements.

One of the first applications of SOP 1.0 was to support the requirements engineering phase, since this phase involves a large amount of information, which can be captured in text form. Its application to the Use Case approach by Cockburn [21] is described in [2]. The approach was also adapted to support the TORE (Task and Object Oriented Requirements Engineering) method, which is described in the following. This application shows that SOP 1.0 can also handle complex document structures.

The *SopTORE* plug-in supports the application of the TORE requirements engineering method in a project context [22]. The method consists of four hierarchical decision-making levels with 16 different types of artifacts, which are used altogether to make decisions explicit during the requirements phase, rather than making these decisions implicit. The main focus of the method lies on the user tasks that the software must support. Since the method is not widespread, this extension supports the TORE method with a set of templates that include specific semantic relations between the requirements artifacts. These relations are based on a documentation model derived from the structure of TORE. This enables the requirements engineering team to see to what degree TORE has been completed inside the Wiki. Through the documentation model, a set of consistency rules can be specified. Furthermore, the extension provides basic project management and rights-management functionality.

The main plug-ins for supporting this document creation are *Textual Template* and *Linking Support*. For each document type mentioned above, a template along with relations to other documents is defined. The consistency rules can be checked with the *Consistency Check* plug-in. Furthermore, the *OpenDoc Export* plug-in was further developed so that ask-statements can be annotated inside the Open Office document. All of the articles present in the result are then exported into the document. After processing the template, all contents of the articles that meet the query are integrated into the document. Finally, the different versioning plug-ins allow defining requirements baselines.

A potential future extension is to retrieve similar documents in order to support linking, in particular across projects. This will increase the reuse of currently existing software artifacts.

## 4.2 Scenario 2: Implementation

During the implementation phase, developers create the software system using IDEs and other relevant tools. During this phase, they implement new methods, make errors in the system, remove them, refactor the system, build the system, debug the system, and (sometimes) test and correct the system.

As it does not make sense to implement proprietary applications, such as Eclipse, and embed them directly into an SOP (we cannot provide the same functionality and stability and developers would not embrace them), we currently use SOP 1.0 to enable and support additional services. Right now, SOP 1.0 does support the implementation phase via:

- Pages that document information about the software system, such as functionality (algorithms / mechanisms) implemented, defects found, treatments (refactorings) applied, or how to deploy or install it
- Pages that document information about variants of the systems, such as specific configurations, releases, or branches (e.g., stored in a version control system)
- Pages that document observations, experiences, and solutions with tools, treatments, or programming languages
- Template-based editing to systematically document bugs (i.e., defect management) as well as test cases, test results, and test data

SOP 1.0 also supports the following aspects supporting implementation:

- Versioning of documented, implementation-specific knowledge.
- Generation of basic reports about features, defects, or (API) changes.
- Traceability support for evaluating dependencies between code and design or requirements.

Future ways of supporting the implementation phase in SOP 2.0 include:

- *Documentation of code*: Code is represented in SOP 2.0 either using un-modifiable pages synchronized with the version control repository (i.e., stored in SVN/CVS and SOP 2.0) or linked dynamically (i.e., stored in SVN/CVS but not in SOP 2.0).
- *Documentation of treatment history*: Defects found by defect detectors such as Findbugs, PMD, or DoctorQ are directly stored in SOP 2.0 and associated with code representations (e.g., a page describing a class). The treatment history (e.g., applied refactorings) is recorded in SOP. This feature would require traceable links

or semantic relationships (e.g., defectFoundIn SVN1::Directory/Class1/Method3::VersionX-Y).

- *Documentation of important algorithms* (e.g., compression algorithm) in pseudo code; Domain knowledge from (multiple) requirement phases. This feature would require traceable links or semantic relationships (e.g., isImplementedIn SVN2::Directory/Class1/Method3::VersionX-Y)
- *Generate Defect Report*: Generate a (dynamic) report about all defects, features, or other changes within a system, version, etc. (including charts, etc.). This feature would require traceable access to all defect data in a specific version and chart-generating extensions.

### 4.3 Scenario 3: Process Modeling

After an initial IESE-internal application describing 100 processes, 60 roles, and about 150 document templates [23], we transferred the SOP 1.0 framework to an organization developing embedded software with 30 software developers. The task was to provide an up-to-date process documentation that could assist in establishing and proving Automotive SPICE<sup>5</sup> compliance, and that provided assistance to the developers in their daily work. In order to utilize and make explicit the knowledge stored in people's heads, it was decided that the documentation contents was to be created by the developers themselves, with a central instance (Software Engineering Process Group, SEPG) setting up structures and coordinating their efforts.

Since calendar time was an issue, a technical solution had to support parallel editing of the resulting process documentation. It was not possible to install special software on all developer computers, so a technical solution could only utilize standard Windows or Linux resources. A third challenge was budget; it was not possible to spend large amounts of money on licenses for software. SOP 1.0 met all these challenges, since it consists only of Open Source software, needs only a Web browser to be used, and provides parallel editing capabilities by its very nature.

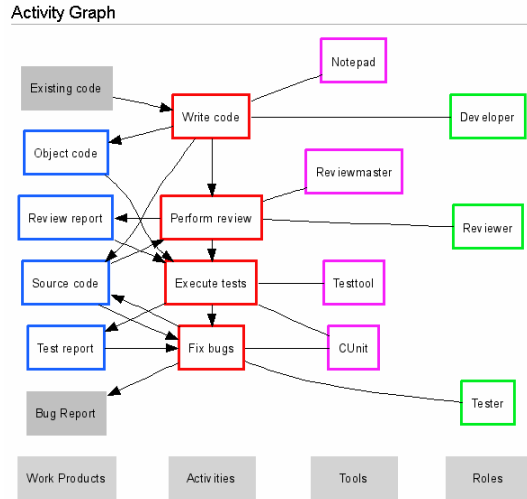


Fig. 3. Representation of process model

<sup>5</sup> <http://www.automotivespice.com/>

The process model within SOP 1.0 is based on four entity (= page) types: processes/activities, work products, tools, and roles. The page structure is predefined using the *Linking Assistance* plug-in, giving users a very simple way to insert new pages that comply with the structure defined by the SEPG. The meta model is very simple: Activities produce and consume artifacts, supported by tools, and have exactly one responsible role and an arbitrary number of contributing or informed roles. Until March 2008, more than 600 pages were created, with more than 3,500 links from one page to another.

In order to keep the resulting process model consistent, automated consistency checks were introduced, based on typed pages and links provided by Semantic MediaWiki. For example, a link of the type “input” was only allowed to point from an activity page to an artifact page, otherwise a consistency error was raised. Many other consistency checks were implemented, e.g., for roles not responsible for or contributing to any activity, activities not having any roles assigned, unused tools, and others. Checking all this manually would have taken days, and the quality of the result would still have been questionable – the automated version only took seconds.

The major extension to the standard SOP 1.0 platform was the visualization of the process model. While syntactic consistency can easily be checked by a machine, semantic consistency remains mostly a human task: *Does it make sense that this activity produces this artifact?* In a pure text-based representation, however, it is very hard to identify such issues (600 pages, 3,500 links), because any reviewer has to construct a mental model of what the text specifies and then needs to examine this mental model. Therefore, we implemented another extension that automatically created graphical representations of the model (parts) specified by the text, again using the semantic information (see Fig. 3). These graphs proved to be an enormous help in assuring semantic consistency, because they always represented the current model and were updated automatically.

Our experiences with using SOP 1.0 as a process modeling platform are very positive. The Wiki syntax proved to be no problem for the editors (mostly software developers). The semantic capabilities were a necessary precondition for using Wiki-technology; without them, creating such process documentation and keeping it consistent would not have been feasible. The open nature of the SOP 1.0 platform allowed us to add extensions wherever necessary.

After using SOP 1.0 for about nine months now, this decision has proven to be the right one. It provides a low-tech, low-budget solution to the challenges raised and allows arbitrary extensions to be added for any new issues arising.

#### **4.4 Scenario 4: Experience Management**

In the context of SOP 1.0, the A2E (Action, Benefit, Context, Description, Evidence) structure [24] was used for gathering, aggregating, and preserving valuable knowledge from old software projects. A2E encompasses all information that is required in the EF. An aggregation technique helps to aggregate observations into experiences and experiences into patterns and laws. The elements used in this structure are described in Table 2.

**Table 2. THE A2E STRUCTURE FOR EXPERIENCES**

	<b>Description</b>
Action	Description of an activity that was applied to cause an outstanding effect.
Benefit	Description of the positive or negative effect that was caused by the action.
Context	Characterization of the environment the action was performed in.
Description	Detailed explanation and depiction of the problem, solution, intent, applicability, etc. of the software pattern – based on a pattern template.
Evidence	Report and list of evidence that back up the claim of the software pattern (e.g., used experiences) as well as other relevant references.

The A2E structure was implemented in SOP 1.0 to structure the knowledge within the EF. The experiences can be annotated with keywords and by choosing context artifacts. A keyword browser supports the user in selecting domain keywords, which come from an OWL ontology based on SWEBOK. These keywords belong to the domain concept categories *product*, *process*, *role*, and *knowledge*. In order to describe the context of an experience (i.e., the (C) of the A2E structure), the SOP 1.0 extension helps to select Wiki pages from the categories *process*, *product*, *process*, *individual*, *group*, *organization*, *customer*, and *software tool*.

#### **4.5 Scenario 5: Learning Spaces**

An approach has been developed to produce so-called learning spaces, first for enhancing the reuse of experience packages and second, for knowledge acquisition. A learning space is generated by the system when a user accesses an experience package (i.e., experience description) from the database during a project. The generation process enriches the experience package with additional situational and instructional content. From a technical point of view, a learning space consists of a hypertext document with linked pages.

A learning space follows a specific global learning goal (the learning goal level is selected by the student) and is created based on context information about the current situation and the experience package. The learning space approach was implemented as an extension of SOP 1.0 and is presented by means of Wiki pages in SOP 1.0. By integrating the learning space generation and presentation functionality into SOP 1.0, knowledge management and e-learning have been merged into one system. See [16] for a detailed description of the concepts in general and the generation process in particular.

## **5 Current Status and Future Work**

The architecture of the upcoming version, SOP 2.0, constitutes fundamental enhancements. It is not only a technological advancement because of a sophisticated framework, but also provides more freedom for developers as well as for users. In addition, it is more tailored to fulfilling the requirements we defined for an SOP.

Right now, we have finished the development of the architecture of SOP 2.0 as depicted in Fig. 2. However, actual experiences came from the usage of SOP 1.0 as described in the scenarios in section 4. Table 3 provides an overview of plug-ins (i.e., features) beneficial for which software development phase (or for experience management and project management associated with every phase).

**Table 3.** FEATURE MATRIX FOR SOP 1.0 AND SOP 2.0<sup>6</sup>

	Features	Requirements	Design	Code	Test	Deployment	Project Management	Experience Management
SOP 1.0	Traceability Matrix	●	●	●	●	○	●	○
	SopTORE	●	○	○	○	○	○	○
	Consistency Check	●	●	◐	●	○	○	○
	Freeze / Unfreeze	●	◐	○	○	○	●	○
	VersionTag List	●	●	●	●	◐	◐	○
	Rights Management (Projects)	○	○	○	○	○	●	○
	Textual Templates	●	●	●	●	●	●	●
	Linking Support	●	●	◐	◐	◐	◐	○
	Learning Spaces	●	●	●	●	●	●	●
	Visual. of Process Models	●	●	●	●	◐	◐	○
	Visual. of Relations	●	●	●	●	◐	◐	○
	OpenDoc Export	●	●	●	●	●	●	●
	SOP 2.0	Document Basket*	●	●	●	●	●	●
Graphical Templates		●	●	●	●	●	●	●
Text Assistance*		●	●	●	●	●	●	●
Visual Browsing*		◐	◐	◐	◐	◐	◐	◐
PID*		●	●	●	●	●	●	●
Expert Management*		●	●	●	●	●	●	●
Personal Desktop*		◐	◐	◐	◐	○	○	○

The feature matrix shows only a subset of the available SOP 1.0 plug-ins, because we abstain from universal plug-ins, such as find & replace in a collection of documents. Features marked with \* are under development or planned.

- *Traceability Matrix* displays typed links between documents (e.g., the links between Use Cases and Actors).
- *SopTORE*: see section 4.1
- *Consistency Check* allows defining SPARQL-based queries that are pre-formed periodically to detect inconsistencies in typed links and metadata.

<sup>6</sup> A filled circle means that the plug-in is entirely appropriate for this phase, whereas a partly-filled circle means that the plug-in is appropriate with exceptions. A white circle stands for no or low application potential.

- *Freeze / Unfreeze* converts links into permalinks and vice versa, thus providing a versioning feature.
- *Version Tag* is another feature that adds a version to multiple documents, selectable via category, metadata, and typed links.
- *Rights Management* allows restricting editing access to documents.
- *Textual Templates and Linking Support*: A user can define templates along with configuring a suggestion for (possible typed) links. For example, for a Use Case template, different types of links to other documents, such as actors, are suggested (e.g., primary, secondary).
- *Learning Spaces* enrich experiences with learning packages (see section 4.5).
- *Visualization of Process Models*: see section 4.3.
- *Visualization of Relations* allows presenting a graphical overview of documents based on metadata and typed links.
- *OpenDoc Export* allows exporting a set of articles as a single document in Open Document Text (ODT) format.
- *Document Basket\**: Consider description below.
- *Graphical Templates*: Flex-UI templates.
- *Text Assistance\**: Similar to eclipse code assist.
- *Visual Browsing\**: Consider description below.
- *PID\**: Consider description below.
- *Expert Management\**: Consider description below.
- *Personal Desktop\**: Similar to Web 2.0 Web Desktops (e.g., Netvibes). It helps to keep a knowledge worker up-to-date and reduces information overload.

Our main objective for the future is to establish SOP 2.0 as the central system within software organizations. This means that SOP 2.0 shall be the central information, knowledge, and experience repository for stakeholders throughout the whole software development process (*RqLife*). In our vision, SOP 2.0 will keep track of all activities throughout the entire software process by being highly integrated into the CASE tool landscape of a software organization. This should be realized by leveraging innovative import and export capabilities or information extraction and aggregation agents, which will help SOP 2.0 to be embraced by all participants (*RqIntg*). These agents will allow seamless integration by recognizing changes (i.e., CRUD (Create, Read, Update, Delete) activities) in external information repositories (e.g., network drives) (*RqOpen*). Extracted information will be automatically analyzed, aggregated, linked, and made accessible within SOP 2.0 (*RqLink*). As an example, the creation of a software component in a CASE tool, such as Eclipse, will be enriched by information extracted from SOP 2.0 in such a sense that the tools access APIs in order to aggregate documentation, results from tests, documented experiences, etc. for this component.

We are currently exploring increased possibilities for more user-friendly and clearly arranged user interfaces, for example with Adobe Flex. This enables graphical applications to visualize relations between documents, roles, or artifacts (*RqSEF*, *RqInfo*). A major problem of the MediaWiki platform is the overview of information structures. In fact, it is easy to create and link documents, but it is hard to preserve the overview, especially if many people create and link documents. Hence, we want to implement a visual navigation application (“Visual Browsing” feature), which should

be as intuitive and suitable as Apple's Coverflow (for browsing graphics) for browsing the complete Wiki documentation graph.

Furthermore, we plan a batch processing functionality, so that a user can collect documents manually or automatically (e.g., by categories), can optionally organize them into tree structures ("Document Basket" feature) and then execute operations on the set of documents (e.g., creation of a report).

Knowledge in software organizations is generally bound to specific persons, for example the creator of software components. An expert management system ("Expert Management" feature) within SOP 2.0 will use the semantic capabilities to link information extracted from versioning systems (e.g., Subversion) with the responsible expert (*RqLink*, *RqInfo*). Expert management will also semantically link persons with documented experiences, further artifacts, and documentation.

Another vision is that SOP 2.0 will provide context-sensitive information through a Personal Information Delivery ("PID" feature) mechanism (*RqInfo*). This mechanism will detect the current context of a user and proactively provide relevant information. The delivered content will be structured and presented according to the needs and preferences of the user. The underlying adaptation (i.e., initiated by the user), respectively adaptivity (i.e., performed automatically by the system without interaction of the user), will be realized by using decision models from the domain of product line engineering. The decision models will explicitly define the commonalities and variabilities of the information structures provided. The different variants will then be instantiated by a resolving algorithm on-demand, i.e., when the user or the system requests context-sensitive information to be presented in SOP 2.0.

## 6 Summary

This article described the concept of a Software Organization Platform (SOP) as the central system for accessing and managing information, knowledge, and experiences in small and medium-sized software organizations. It presented recent experiences, the current status, and future perspectives of an implementation of SOP. First experiences in industrial and research projects have shown that an SOP is an effective, easy-to-use, extendable system, which can handle a broad variety of information and is able to support various business tasks within a software organization.

Nevertheless, the initial prototype of an SOP (called SOP 1.0) covered only specific tasks of a software process: The upcoming SOP 2.0 and the popular Web 2.0 technologies could serve as the basis for creating new applications that provide easy access to up-to-date information from various sources. The new architecture of SOP 2.0 has the potential of fulfilling most of the requirements defined for an SOP and will provide extended flexibility and extendibility. More user-friendly and intuitive user interfaces, innovative services, personal information delivery, interfaces to third-party systems, or easy creation and customization of mashup applications within an SOP will provide knowledge workers in software organizations with innovative possibilities for managing information and knowledge on personal, team, and organizational levels.

The semantic platform that forms the basis for the implementation of SOP 2.0 provides promising new ways of storing and retrieving information within software projects. It enables the system to deliver information and experiences in context-sensitive and personally adapted ways. Contrary to complex, expensive, and formal systems, an SOP can provide both, low-threshold technologies for storing and accessing information and the possibilities of machine-readable formats and semantic Web technologies.

**Acknowledgements** We acknowledge proof-reading of this article by Savitha Chennagiri and Sonnhild Namingha.

## References

1. Rus, I., Lindvall, M.: Knowledge Management in Software Engineering. *IEEE Software*, vol. 19, no. 3 (2002)
2. Decker, B., Ras, E., Rech, J., Jaubert, P., Rieth, M.: Wiki-based Stakeholder Participation in Requirements Engineering. *IEEE Software*, 24(2), pp. 28--35 (2007)
3. Oddy, G.C.: Software engineering environments. *CompEuro '88. Design, Concepts, Methods and Tools*, pp. 362-367 (1988)
4. Skuce, D.: Knowledge Management in Software Design. A Tool and a Trial. *Software Engineering Journal*, no. 5, vol. 10, pp. 183--193 (1995)
5. Lethbridge, T: Tim Lethbridge's PhD Research, <http://www.site.uottawa.ca/~tcl/personal/PhDWork.shtml>
6. Henniger, S.: Case-Based Knowledge Management Tools in Software Development. *Automated Software Engineering*, no. 3, vol. 4, pp. 319--339 (1997)
7. O'Reilly, T.: Different Approaches to the Semantic Web, <http://radar.oreilly.com/archives/2007/03/different-approaches-to-the-se.html>
8. Khare, R.: Microformats. The next (small) thing on the semantic web? *IEEE internet computing*, vol. 10, no. 1, pp. 68--75 (2006)
9. Basili, V.R., Caldiera, G., Rombach, H.D.: Experience Factory. In: Marciniak, J.J. (ed.): *Encyclopedia of Software Engineering*, vol. 1. John Wiley & Sons, New York, pp. 469--476 (1994)
10. Frost, R.: Jazz and the eclipse way of collaboration. *IEEE software*, vol. 24, no. 6, pp. 114--117 (2007)
11. Versteegen, G.: Anforderungsgetriebenes ALM. *Objektspekturm. Online-ausgabe requirements engineering* (2007)
12. Sharon, D.; Anderson, T.: A complete software engineering environment. *IEEE software*, vol.14, no.2, pp. 123--125 (1997)
13. Chao, J.: Student Project Collaboration Using Wikis. *20th Conference on Software Engineering Education & Training, CSEET '07*, pp. 255--261 (2007)
14. Wikipedia, <http://wikipedia.com>
15. Trac open source project, <http://trac.edgewall.org>
16. Rech, J., Ras, E., & Decker, B.: Riki. A System for Knowledge Transfer and Reuse in Software Engineering Projects. In: M. Lytras & A. Naeve (Eds.): *Open Source for Knowledge and Learning Management: Strategies beyond Tools*. Idea Group, Inc. (2007)
17. Greaves, M.: Semantic Web 2.0. In: *IEEE Intelligent Systems*, vol. 22, no. 2, March-April, pp. 94--96 (2007)
18. Web 2.0: Compact Definition?, [http://radar.oreilly.com/archives/2005/10/web\\_20\\_compact\\_definition.html](http://radar.oreilly.com/archives/2005/10/web_20_compact_definition.html)

19. Segaran, T.: Programming Collective Intelligence, O'Reilly, p. 2 (2007)
20. Rech, J., Ras, E., Decker, B.: Intelligent Assistance in German Software Development. A Survey. IEEE Software, July-Aug, vol. 24, no. 4, pp. 72--79 (2007)
21. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley Longman Publishing Co. Inc. (2000)
22. Paech, B.; Kohler, K.: Task-Driven Requirements in Object-Oriented Development. In: Leite, J.C.S.P. (Ed.), Doorn, J.H. (Ed.): Perspectives on Software Requirements. Dordrecht Kluwer, Academic Publishers, pp. 45--67 (2004)
23. Decker, B.: Using Semantic Wiki Technology for Collaborative Software Process Evolution Workshop on Learning Software Organization, Co-located with Conference for Professional knowledge Management, Potsdam (2007)
24. Rech, J., Ras, E.: Aggregation von Erfahrungen in Erfahrungsdatenbanken. Künstliche Intelligenz, p. 6. (2007)